



Faculty of Engineering & Technology
Electrical & Computer Engineering Department
ARTIFICIAL INTELLIGENCE

ENCS3340

Report for AI PROJECT

Student 1: Salwa Fayyad 1200430

Student 2: Sondos Ashraf 1200905

Instructor: Dr. Yazan Abu Farha

Section: 2

Date: 20.6.2023

Table of Contents

A. About our program	3
B. Heuristic Function	4
C. Result Explain:	5

A. About our program

The provided code is a Java implementation of a game called Magnetic Cave. The game is played on an 8x8 board, where two players take turns placing their bricks on empty cells. The goal is to create a sequence of five bricks either horizontally, vertically, or diagonally.

The game supports two modes: playing against an AI computer or playing against another player. The AI computer utilizes the minimax algorithm with alpha-beta pruning to make intelligent moves. The depth of the algorithm determines the lookahead of the AI.

The code includes methods for initializing the board, displaying the board, validating moves, checking for a win, making moves, evaluating the board state, and switching players. The `evaluateBoard()` method calculates a score based on the number of bricks in rows, columns, and diagonals. The score is used by the AI to determine the best move.

The `play()` method orchestrates the gameplay. It prompts the user to select the game mode and the starting player. Players make moves alternately until a win or tie is detected. The game ends with a display of the final board and the winner.

Overall, the code provides a functional implementation of the Magnetic Cave game with an AI opponent. It demonstrates the use of algorithms and techniques for creating a game-playing AI.

B. Heuristic Function

The heuristic we used in our code is a simple evaluation function that assigns scores to different board positions based on the presence of the player's bricks and the opponent's bricks. The heuristic aims to evaluate the strength of a given board position for the current player. The evaluation function consists of three main components: evaluating rows, evaluating columns, and evaluating diagonals.

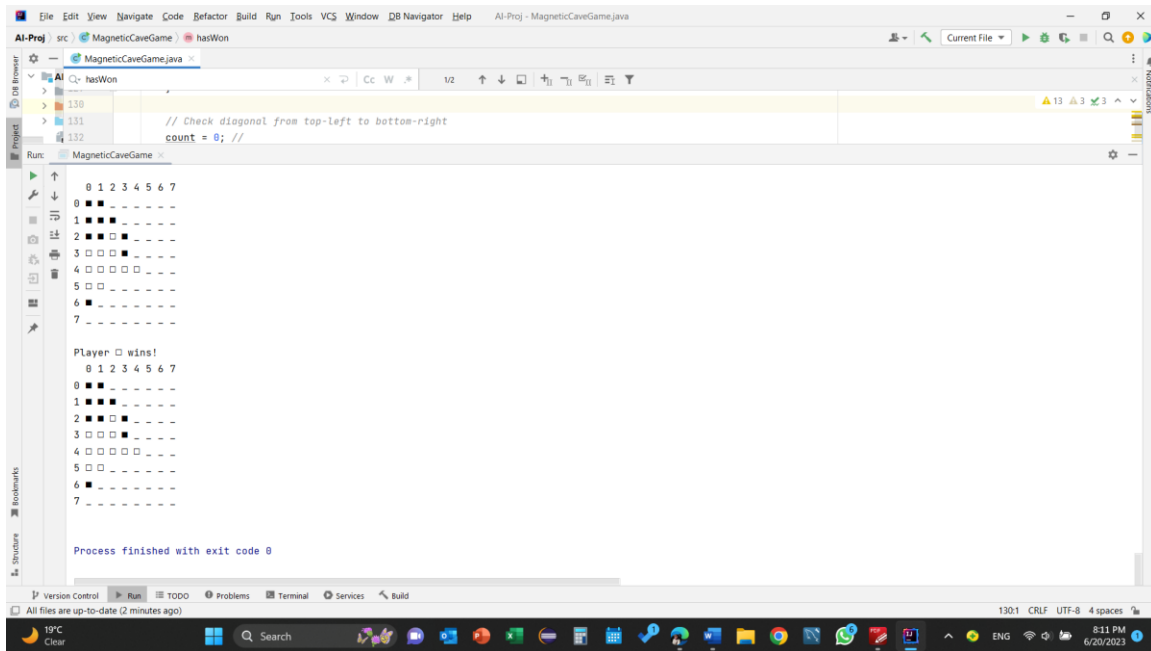
1. **Evaluating Rows:** The code iterates through each row and checks for sequences of bricks. It assigns a score to the row based on the number of bricks owned by the player and the opponent. If the player has more bricks in a sequence, the score is decreased to penalize the player. Conversely, if the opponent has more bricks, the score is increased to reward the player.
2. **Evaluating Columns:** Similar to evaluating rows, the code iterates through each column and assigns a score based on the number of player and opponent bricks in each column.
3. **Evaluating Diagonals:** The code evaluates two types of diagonals: top-left to bottom-right and top-right to bottom-left. It assigns scores based on the presence of player and opponent bricks in each diagonal.

While this heuristic is relatively simple, it can guide the AI player towards making reasonable moves that prioritize the formation of sequences of bricks and blocking the opponent. However, it may not capture all possible winning or losing positions, as it only evaluates the current state of the board and does not consider future moves beyond the specified depth in the minimax algorithm.

To improve the AI player's performance, more advanced heuristics can be developed, considering additional patterns and strategies. Additionally, increasing the depth of the minimax algorithm can allow the AI to look further ahead and make more informed decisions

C. Result Explain:

We play the game as manual entry for ■'s moves & automatic moves for □ mode. As you can see in figure below the □ win because he followed minimax algorithm to know the best move to win. Note that one player is able to align 5 consecutive bricks in a row, in a column or in a diagonal, then this player wins the game. The opposite player lose maybe because he didn't played by minimax algorithm or he cared about losing the against player instead of care to win.



```
File Edit View Navigate Code Refactor Build Run Tools VCS Window DB Navigator Help AI-Proj - MagneticCaveGame.java
AI-Proj src MagneticCaveGame hasWon
MagneticCaveGame.java
130
131 // Check diagonal from top-left to bottom-right
132 count = 0; //

Run: MagneticCaveGame
0 1 2 3 4 5 6 7
0 ■ ■ ■ ■ ■ ■ ■
1 ■ ■ ■ ■ ■ ■ ■
2 ■ ■ ■ ■ ■ ■ ■
3 □ □ □ □ □ □ □
4 □ □ □ □ □ □ □
5 □ □ □ □ □ □ □
6 ■ ■ ■ ■ ■ ■ ■
7 ■ ■ ■ ■ ■ ■ ■

Player □ wins!
0 1 2 3 4 5 6 7
0 ■ ■ ■ ■ ■ ■ ■
1 ■ ■ ■ ■ ■ ■ ■
2 ■ ■ ■ ■ ■ ■ ■
3 □ □ □ □ □ □ □
4 □ □ □ □ □ □ □
5 □ □ □ □ □ □ □
6 ■ ■ ■ ■ ■ ■ ■
7 ■ ■ ■ ■ ■ ■ ■

Process finished with exit code 0
```